

5

Writing Control Structures

ORACLE®

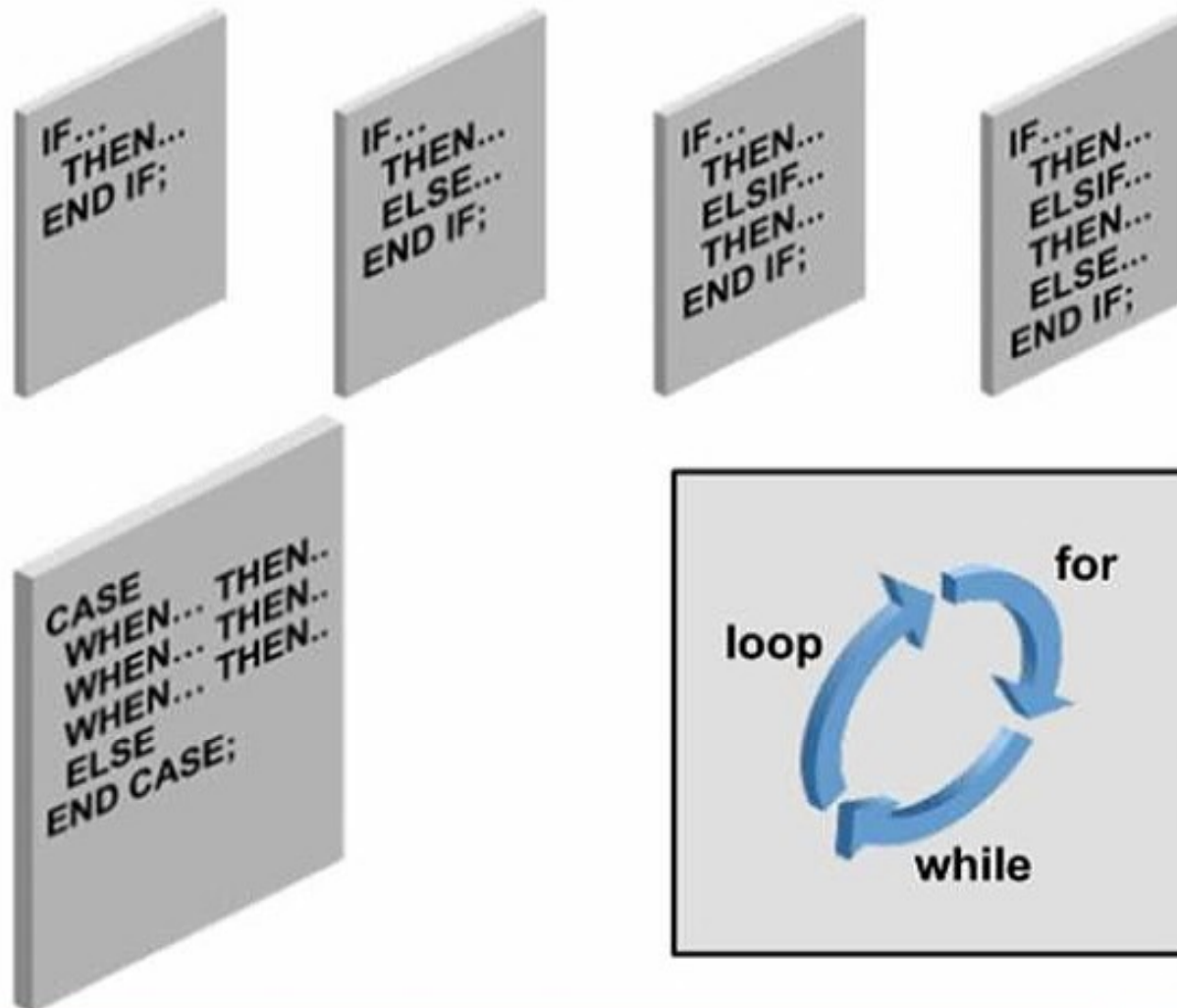
Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Identify the uses and types of control structures
- Construct an `IF` statement
- Use `CASE` statements and `CASE` expressions
- Construct and identify loop statements
- Use guidelines when using conditional control structures

Controlling Flow of Execution



IF Statement

Syntax:

```
IF condition THEN  
    statements;  
[ELSIF condition THEN  
    statements;]  
[ELSE  
    statements;]  
END IF;
```

Simple IF Statement

```
DECLARE
  v_myage  number:=31;
BEGIN
  IF v_myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  END IF;
END;
/
```

anonymous block completed

IF THEN ELSE Statement

```
DECLARE
v_myage  number:=31;
BEGIN
IF v_myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

IF ELSIF ELSE Clause

```
DECLARE
  v_myage number:=31;
BEGIN
  IF v_myage < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSIF v_myage < 20 THEN
    DBMS_OUTPUT.PUT_LINE(' I am young ');
  ELSIF v_myage < 30 THEN
    DBMS_OUTPUT.PUT_LINE(' I am in my twenties');
  ELSIF v_myage < 40 THEN
    DBMS_OUTPUT.PUT_LINE(' I am in my thirties');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am always young ');
  END IF;
END;
/
```

```
anonymous block completed
I am in my thirties
```

NULL Value in IF Statement

```
DECLARE
  v_myage  number;
BEGIN
  IF v_myage < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
/
```

```
anonymous block completed
I am not a child
```


CASE Expressions

- A CASE expression selects a result and returns it.
- To select the result, the CASE expression uses expressions. The value returned by these expressions is used to select one of several alternatives.

```
CASE selector
  WHEN expression1 THEN result1
  WHEN expression2 THEN result2
  ...
  WHEN expressionN THEN resultN
  [ELSE resultN+1]
END;
/
```

CASE Expressions: Example

```
SET VERIFY OFF
DECLARE
    v_grade CHAR(1) := UPPER('&grade');
    appraisal VARCHAR2(20);
BEGIN
    appraisal := CASE v_grade
        WHEN 'A' THEN 'Excellent'
        WHEN 'B' THEN 'Very Good'
        WHEN 'C' THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || '
                          Appraisal ' || appraisal);
END;
/
```

Searched CASE Expressions

```
DECLARE
  v_grade CHAR(1) := UPPER('&grade');
  appraisal VARCHAR2(20);
BEGIN
  appraisal := CASE
    WHEN v_grade = 'A' THEN 'Excellent'
    WHEN v_grade IN ('B','C') THEN 'Good'
    ELSE 'No such grade'
  END;
  DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || '
                        Appraisal ' || appraisal);
END;
/
```

CASE Statement

```
DECLARE
  v_deptid NUMBER;
  v_deptname VARCHAR2(20);
  v_emps NUMBER;
  v_mngid NUMBER:= 108;
BEGIN
  CASE v_mngid
    WHEN 108 THEN
      SELECT department_id, department_name
        INTO v_deptid, v_deptname FROM departments
        WHERE manager_id=108;
      SELECT count(*) INTO v_emps FROM employees
        WHERE department_id=v_deptid;
    WHEN 200 THEN
      ...
    END CASE;
  DBMS_OUTPUT.PUT_LINE ('You are working in the ' || deptname ||
    ' department. There are ' || v_emps || ' employees in this
    department');
END;
/
```

Handling Nulls

When working with nulls, you can avoid some common mistakes by keeping in mind the following rules:

- Simple comparisons involving nulls always yield `NULL`.
- Applying the logical operator `NOT` to a null yields `NULL`.
- If the condition yields `NULL` in conditional control statements, its associated sequence of statements is not executed.

Logic Tables

Build a simple Boolean condition with a comparison operator.

AND	<i>TRUE</i>	<i>FALSE</i>	<i>NULL</i>	OR	<i>TRUE</i>	<i>FALSE</i>	<i>NULL</i>	NOT	
<i>TRUE</i>	TRUE	FALSE	NULL	<i>TRUE</i>	TRUE	TRUE	TRUE	<i>TRUE</i>	FALSE
<i>FALSE</i>	FALSE	FALSE	FALSE	<i>FALSE</i>	TRUE	FALSE	NULL	<i>FALSE</i>	TRUE
<i>NULL</i>	NULL	FALSE	NULL	<i>NULL</i>	TRUE	NULL	NULL	<i>NULL</i>	NULL

Boolean Conditions

What is the value of `flag` in each case?

```
flag := reorder_flag AND available_flag;
```

REORDER_FLAG	AVAILABLE_FLAG	FLAG
TRUE	TRUE	? (1)
TRUE	FALSE	? (2)
NULL	TRUE	? (3)
NULL	FALSE	? (4)

Iterative Control: LOOP Statements

- Loops repeat a statement (or sequence of statements) multiple times.
- There are three loop types:
 - Basic loop
 - FOR loop
 - WHILE loop



Basic Loops

Syntax:

```
LOOP  
  statement1;  
  . . .  
  EXIT [WHEN condition];  
END LOOP;
```

Basic Loops

Example:

```
DECLARE
  v_countryid    locations.country_id%TYPE := 'CA';
  v_loc_id       locations.location_id%TYPE;
  v_counter      NUMBER(2) := 1;
  v_new_city     locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
/
```

WHILE Loops

Syntax:

```
WHILE condition LOOP  
    statement1;  
    statement2;  
    . . .  
END LOOP;
```

Use the `WHILE` loop to repeat statements while a condition is `TRUE`.

WHILE Loops: Example

```
DECLARE
  v_countryid  locations.country_id%TYPE := 'CA';
  v_loc_id     locations.location_id%TYPE;
  v_new_city   locations.city%TYPE := 'Montreal';
  v_counter    NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
  END LOOP;
END;
/
```

FOR Loops

- Use a `FOR` loop to shortcut the test for the number of iterations.
- Do not declare the counter; it is declared implicitly.

```
FOR counter IN [REVERSE]  
    lower_bound..upper_bound LOOP  
    statement1;  
    statement2;  
    . . .  
END LOOP;
```

FOR Loops: Example

```
DECLARE
  v_countryid  locations.country_id%TYPE := 'CA';
  v_loc_id     locations.location_id%TYPE;
  v_new_city   locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id
    FROM locations
   WHERE country_id = v_countryid;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
      VALUES((v_loc_id + i), v_new_city, v_countryid );
  END LOOP;
END;
/
```

FOR Loops

Guidelines

- Reference the counter within the loop only; it is undefined outside the loop.
- Do not reference the counter as the target of an assignment.
- Neither loop bound should be `NULL`.

Guidelines for Loops

- Use the basic loop when the statements inside the loop must execute at least once.
- Use the `WHILE` loop if the condition must be evaluated at the start of each iteration.
- Use a `FOR` loop if the number of iterations is known.

Nested Loops and Labels

- You can nest loops to multiple levels.
- Use labels to distinguish between blocks and loops.
- Exit the outer loop with the `EXIT` statement that references the label.

Nested Loops and Labels

```
...  
BEGIN  
  <<Outer_loop>>  
  LOOP  
    v_counter := v_counter+1;  
    EXIT WHEN v_counter>10;  
    <<Inner_loop>>  
    LOOP  
      ...  
      EXIT Outer_loop WHEN total_done = 'YES';  
      -- Leave both loops  
      EXIT WHEN inner_done = 'YES';  
      -- Leave inner loop only  
      ...  
    END LOOP Inner_loop;  
    ...  
  END LOOP Outer_loop;  
END;  
/
```


PL/SQL CONTINUE Statement

- Definition
 - Adds the functionality to begin the next loop iteration
 - Provides programmers with the ability to transfer control to the next iteration of a loop
 - Uses parallel structure and semantics to the `EXIT` statement
- Benefits
 - Eases the programming process
 - May see a small performance improvement over the previous programming workarounds to simulate the `CONTINUE` statement



PL/SQL CONTINUE Statement: Example

```
DECLARE
  v_total SIMPLE_INTEGER := 0;
BEGIN
  FOR i IN 1..10 LOOP
    ① v_total := v_total + i;
      dbms_output.put_line
        ('Total is: ' || v_total);
      CONTINUE WHEN i > 5;
    ② v_total := v_total + i;
      dbms_output.put_line
        ('Out of Loop Total is:
         ' || v_total);
    END LOOP;
  END;
  /
```



```
anonymous block completed
Total is: 1
Out of Loop Total is:
  2
Total is: 4
Out of Loop Total is:
  6
Total is: 9
Out of Loop Total is:
 12
Total is: 16
Out of Loop Total is:
 20
Total is: 25
Out of Loop Total is:
 30
Total is: 36
Total is: 43
Total is: 51
Total is: 60
Total is: 70
```

PL/SQL CONTINUE Statement: Example

```
DECLARE
  v_total NUMBER := 0;
BEGIN
  <<BeforeTopLoop>>
  FOR i IN 1..10 LOOP
    v_total := v_total + 1;
    dbms_output.put_line
      ('Total is: ' || v_total);
    FOR j IN 1..10 LOOP
      CONTINUE BeforeTopLoop WHEN i + j > 5;
      v_total := v_total + 1;
    END LOOP;
  END LOOP;
END two_loop;
```

```
anonymous block completed
Total is: 1
Total is: 6
Total is: 10
Total is: 13
Total is: 15
Total is: 16
Total is: 17
Total is: 18
Total is: 19
Total is: 20
```

Quiz

There are three types of loops: Basic, FOR, and WHILE.

1. True
2. False

Summary

In this lesson, you should have learned how to change the logical flow of statements by using the following control structures:

- Conditional (`IF` statement)
- `CASE` expressions and `CASE` statements
- Loops:
 - Basic loop
 - `FOR` loop
 - `WHILE` loop
- `EXIT` statement
- `CONTINUE` statement